

## AN EFFICIENT SCHEMA FOR STORING MULTI-VALUE ATTRIBUTES IN A DIRECTORY SERVICE BACKING STORE

This application includes subject matter protected by  
5 copyright. All rights are reserved.

### BACKGROUND OF THE INVENTION

#### Technical Field

This invention relates generally to providing directory  
services in a distributed computing environment.

#### 10 Description of the Related Art

A directory service is the central point where network  
services, security services and applications can form an  
integrated distributed computing environment. Typical uses  
of a directory services may be classified into several  
15 categories. A "naming service" (e.g., DNS and DCE Cell  
Directory Service (CDS)) uses the directory as a source to  
locate an Internet host address or the location of a given  
server. A "user registry" (e.g., Novell NDS) stores  
information about users in a system composed of a number of  
20 interconnected machines. The central repository of user  
information enables a system administrator to administer the  
distributed system as a single system image. Still another  
directory service is a "white pages" lookup provided by some  
e-mail clients, e.g., Netscape Communicator, Lotus Notes,  
25 Endora and the like).

With more and more applications and system services  
demanding a central information repository, the next

generation directory service will need to provide system administrators with a data repository that can significantly ease administrative burdens. In addition, the future directory service must also provide end users with a rich information data warehouse that allows them to access department or company employee data, as well as resource information, such as name and location of printers, copy machines, and other environment resources. In the Internet/intranet environment, it will be required to provide user access to such information in a secure manner.

To this end, the Lightweight Directory Access Protocol (LDAP) has emerged as an IETF open standard to provide directory services to applications ranging from e-mail systems to distributed system management tools. LDAP is an evolving protocol that is based on a client-server model in which a client makes a TCP/IP connection to an LDAP server, sends requests, and receives responses. The LDAP information model in particular is based on an "entry," which contains information about some object. Entries are typically organized in a specified tree structure, and each entry is composed of attributes.

LDAP provides a number of known functions including query (search and compare), update, authentication and others. The search and compare operations are used to retrieve information from the database. For the search

function, the criteria of the search is specified in a search filter. The search filter typically is a Boolean expression that consists of qualifiers including attribute name, attribute value and Boolean operators like AND, OR and  
5 NOT. Users can use the filter to perform complex search operations. One filter syntax is defined in RFC 2254.

LDAP thus provides the capability for directory information to be efficiently queried or updated. It offers a rich set of searching capabilities with which users can  
10 put together complex queries to get desired information from a backing store. Increasingly, it has become desirable to use a relational database for storing LDAP directory data. Representative database implementations include DB/2, Oracle, Sybase, Informix and the like. As is well known,  
15 Structured Query Language (SQL) is the standard language used to access such databases.

Relational database guidelines (e.g., the First Normal Form) requires that attributes within each tuple are ordered and complete and that searchable domains permit only simple  
20 values. Further, simple values cannot be decomposed into multiple values, and they cannot be decomposed into multiple relations. If these guidelines are not followed, the database application becomes quite difficult to manage. Such limitations present difficulties when it is desired to  
25 use a relational database as an LDAP backing store. In

particular, LDAP allows multi-value attributes. As a result, implementation of the LDAP directory model requires that there is a relation (or table) for each searchable attribute. This "per attribute" table design basically  
5 normalizes all the attributes to achieve a manageable implementation. A schema of this type provides a general solution for LDAP applications.

However, for applications which rarely use multi-value attributes, the per attribute table does not perform well  
10 for certain functions. In particular, add/updates are very expensive. Thus, for example, for an entry with ten attributes, more than ten tables need to be updated. As another example, logical operations involving multiple attributes require expensive table joins to perform the  
15 operation.

It would be highly desirable to provide a database schema to solve the performance problem of per-attribute tables, especially for directory applications that rarely use multi-value attributes. The present invention solves  
20 this problem.

**BRIEF SUMMARY OF THE INVENTION**

It is a primary object of this invention to provide a flexible and efficient database schema for a directory service having a relational database backing store.

5 It is another primary object of this invention to solve the performance problem of per-attribute tables, especially for directory applications that rarely use multi-value attributes.

10 It is yet another primary object of the present invention to provide an efficient method of dealing with multi-value attributes in a directory service, e.g., a service conforming to the Lightweight Directory Access Protocol ("LDAP").

15 A still further object of this invention is to extend the LDAP attribute schema to facilitate handling of multi-value attributes in a relational database backing store.

A more general object of this invention is to provide a reliable and scaleable enterprise directory solution, wherein a preferred implementation is LDAP using a DB/2  
20 backing store.

These and other objects of the invention are achieved by a database schema that includes a so-called "merged" attribute table. The merged attribute table stores normalized attributes to facilitate database searches.  
25 Generally, the merged attribute table stores single value

attributes, wherein multi-value attributes are stored in the per attribute tables, however, each attribute can exist either in the merged table, an attribute table or both.

In a preferred embodiment, the present invention  
5 describes a database schema for storing application data in a backing store of a directory service. Thus, for example, the directory service is LDAP and the backing store is a relational database, such as DB2. The application data has at least some entries with multiple value attributes.  
10 According to the invention, the application data is profiled to determine how it may be optimally stored in the backing store. Preferably, single entries having single value attributes are stored in a merged attribute table, while entries having multiple value attributes are stored in per  
15 attribute tables. According to the optimization, a majority of the attributes are single valued and are stored in the merged table, and the per attribute tables thus store a relatively smaller number of exceptions. This database schema enhances processing of conventional directory service  
20 queries into the backing store.

The foregoing has outlined some of the more pertinent objects and features of the present invention. These objects and features should be construed to be merely illustrative of some of the more prominent features and  
25 applications of the invention. Many other beneficial

results can be attained by applying the disclosed invention in a different manner or modifying the invention as will be described. Accordingly, other objects and a fuller understanding of the invention may be had by referring to  
5 the following Detailed Description of the preferred embodiment.

006372.00197:0416317.01

**BRIEF DESCRIPTION OF THE DRAWINGS**

For a more complete understanding of the present invention and the advantages thereof, reference should be made to the following Detailed Description taken in  
5 connection with the accompanying drawings in which:

**Figure 1** is a representative LDAP directory service implementation;

**Figure 2** is a simplified LDAP directory;

**Figure 3** is a flowchart of an LDAP directory session;

10 **Figures 4A-4B** show representative LDAP directory service implementations having a relational database backing store;

**Figure 5** is a simplified diagram of the inventive database schema;

15 **Figure 6** is a diagram illustrating a process for profiling an application data set to identify appropriate entries for the merged table;

**Figure 7** is a simplified flowchart of the data profiling process; and

20 **Figure 8** is a simplified representation of the merged table structure.



### DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A block diagram of a representative LDAP directory service in which the present invention may be implemented is shown in **Figure 1**. As is well-known, LDAP is the  
5 lightweight directory access protocol, and this protocol has been implemented in the prior art, e.g., as either a front end to the X.500 directory service, or as a standalone directory service. According to the protocol, a client machine **10** makes a TCP/IP connection to an LDAP server **12**,  
10 sends requests and receives responses. LDAP server **12** supports a directory **21** as illustrated in a simplified form in **Figure 2**. Each of the client and server machines further include a directory "runtime" component **25** for implementing the directory service operations as is described below. The  
15 directory **21** is based on the concept of an "entry" **27**, which contains information about some object (e.g., a person). Entries are composed of attributes **29**, which have a type and one or more values. Each attribute **29** has a particular syntax that determines what kinds of values are allowed in  
20 the attribute (e.g., ASCII characters, .jpeg file, etc.) and how these values are constrained during a particular directory operation.

The directory tree is organized in a predetermined manner, with each entry uniquely named relative to its  
25 sibling entries by a "relative distinguished name" (RDN).

An RDN comprises at least one distinguished attribute value from the entry and, at most, one value from each attribute is used in the RDN. According to the protocol, a globally unique name for an entry, referred to as a "distinguished name" (DN), comprises a concatenation of the RDN sequence  
5 from a given entry to the tree root.

The LDAP search can be applied to a single entry (a base level search), an entry's children (a one level search), or an entire subtree (a subtree search). Thus, the  
10 "scope" supported by LDAP search are: base, one level and subtree. LDAP does not support search for arbitrary tree levels and path enumeration.

LDAP includes an application programming interface (API), as described in "The C LDAP Application Program  
15 Interface", IETF Working Draft, July 29, 1997, which is incorporated herein by reference. An application on a given client machine uses the LDAP API to effect a directory service "session" according to the flowchart of **Figure 3**. At step **40**, an LDAP session with a default LDAP server is  
20 initialized. At step **42**, an API function `ldap_init()` returns a handle to the client, and this handle may allow multiple connections to be open at one time. At step **44**, the client authenticates to the LDAP server using, for example, an API `ldap_bind()` function. At step **46**, one or  
25 more LDAP operations are performed. For example, the API

function ldap\_search() may be used to perform a given directory search. At step 48, the LDAP server returns the results of the directory search, e.g., one or more database elements that meet the search criteria. The session is then  
5 closed at step 50 with the API ldap\_unbind() function then being used to close the connection.

It may be desirable to store LDAP directory data in a backing store. **Figures 4A-4B** illustrate several representative LDAP directory service implementations that  
10 use a relational database management system (RDBMS) for this purpose. These systems merely illustrate possible LDAP directory services in which the present invention may be implemented. One of ordinary skill should appreciate, however, that the invention is not limited to an LDAP  
15 directory service provided with a DB/2 backing store. The principles of the present invention may be practiced in other types of directory services (e.g., X.500) and using other relational database management systems (e.g., Oracle, Sybase, Informix, and the like) as the backing store.

20 In **Figure 4A**, an LDAP client 34 can connect to a number of networked databases 38a-58n through an LDAP server 36. The databases 38a-38n contain the directory information. However, from the user's perspective, the LDAP server 36 stores all the information without knowing the database 38  
25 in which the data is actually located. With this

configuration, the LDAP server 36 is freed from managing the physical data storage and is able to retrieve information from multiple database servers 38 which work together to form a huge data storage.

5       **Figure 4B** illustrates a multiple client/multiple server LDAP/DB2 enterprise solution. In this environment, a DB/2 client preferably runs on each LDAP server 36. Each such DB/2 client can connect to any database server 38 containing directory information. The collection of database servers  
10   38a-38n form a single directory system image, and one or more of the LDAP servers 36 can access such information. Because all the LDAP servers 36 see the same directory image, a network dispatcher 37 may be deployed to route requests among the LDAP servers 36.

15       One of ordinary skill should appreciate that the system architectures illustrated in **Figures 4A-4B** are not to be taken as limiting the present invention. The inventive technique may be used to search any relational database using hierarchical, filter-based database queries.

20       Implementation of the LDAP directory model requires that there is a relation (or table) for each searchable attribute. This is sometimes referred to as a "per attribute" table database schema. For applications that rarely use multi-value attributes, the per attribute table  
25 does not perform well for certain operations. The present

invention solves this problem by providing a new database schema wherein a so-called "merged" attribute table is used to store single value attributes. A simplified illustration of the inventive schema is shown in **Figure 5**.

5       As illustrated in this drawing, the novel schema comprises the merged attribute table **60** and the set of per attribute tables **62a-62n**. A given set of per attribute tables is sometimes referred to herein as an "overflow" table. According to the present invention, single value  
10 attributes **64** are stored into the merged table **60**, and multiple value attributes **66** are typically stored in the per attribute tables **62**. As will be seen, however, each attribute can exist in the merged table, the per attribute tables, or both.

15       In the LDAP Version 3.0 schema, attributes having a single value are specified by a SINGLE-VALUE keyword. For example, the attribute "c" contains a two-letter ISO 3166 country code (countryName). The schema definition of "c" is then as follows: (2.5.4.6 NAME 'c' SUP name SINGLE-VALUE).  
20 For attributes **66** that are not specified as a single-value attribute in the schema, according to the invention, the default is a multi-value attribute.

In certain circumstances, it may not be desirable to place all multi-value attributes in the per attribute  
25 tables. For example, even though the attribute is specified

as a multi-value attribute in the standard schema, the user application may only have a single value in the entry. Another problematic situation arises when the attribute is truly multi-valued, yet a significant number of entries contain more than one value for the attribute. Thus, according to the present invention, it is desirable to parse the application data to be stored in the backing store to determine an optimal configuration for populating the merged attribute table. This process is illustrated in the block diagram of **Figure 6**.

As illustrated in this figure, application data to be stored in the backing store is first profiled by a parsing routine 70. The parsing routine may be any given data analysis tool, e.g., an LDIF file, a Lotus Notes NAB or a collection of database records. The output of the parsing routine is a file 72 that preferably comprises the following information for each attribute:

- Attribute name
- Maximum length of the attribute
- Total number of entries that contain this attribute (TOT\_E)
- Number of entries that contain multiple values of the target attribute (TOT\_MULTI)
- Percentage entries that contain multiple values of the target attribute ( $TOT\_PERCENT = TOT\_MULTI/TOT\_E$ )

The following is an example of the output file 72 generated from the ldif file of representative directory data:

Attribute Name: Name  
Max Length: 30

Total number of entries (TOT\_E): 10,000  
Multi-Value entries (TOT\_MULTI): 0  
Percentage: 0%

5 Attribute Name: Office Phone  
Max Length: 12  
Total number of entries (TOT\_E): 10,000  
Multi-Value entries (TOT\_MULTI): 100  
Percentage: 1%

10 Attribute Name: Manager  
Max Length: 30  
Total number of entries (TOT\_E): 10,000  
Multi-Value entries (TOT\_MULTI): 500

15 Percentage: 2%

In the example above, it can be seen that only a relatively small number of entries have multiple values for the attributes OFFICE PHONE and MANAGER, while the NAME attribute has none (in this example). In other words, all of the people in the representative directory have only one name.

The output of the parsing routine 70 is then used to instruct the directory service as to the optimal way to organize the data stored in the database. In particular, after the data profiling output file 72 is generated, a schema generator 74 uses the profiling output to generate the schema file 75 for the LDAP server. With the example above, the following are representative attribute definitions in the schema file 75:

(2.5.4.4 NAME 'NAME' MERGETBL (merged \$ Name)  
INDEX (F))

(2.5.4.10 NAME 'PHONE' MERGETBL (merged \$ Phone)  
ATTRTBL (Phone \$ Phone)  
INDEX (F))

(2.5.4.4 NAME 'NAME' MERGETBL (merged \$ Manager)  
ATTRTBL (Manager \$ Manager)  
INDEX (F))

5

As will be described, the single value attributes (NAME, for example) preferably are stored in the merged table. For those attributes that have a small number of multiple values (OFFICE PHONE and MANAGER, for example), they are stored both in the merged table and the per attribute table. The schema file 75 generated by the schema generator 74 is used by the LDAP server daemon.

Figure 7 is a simplified high level flowchart of the application data parsing routine. The routine begins at step 76 by analyzing the application data. At step 78, the data is profiled to determine whether the data should be in stored in an attribute table or, alternatively, in a merged table and an overflow table comprising a set of attribute tables. The profiling step, for example, parses the data to identify entries with single value attributes. The profiling step may also parse the data to identify given operations that are performed on the data once stored.

At step 80, the routine determine the optimum storage of the data based on the profile. At step 82, given data is stored in an attribute table. At step 84, other given data is stored in a merged table. As noted above, preferably the merged table is used for entries with single value attributes. Preferably, a majority of the data is stored in



the merged table and a small set of additional values for the multiple value attributes are stored in the per attribute tables.

The following is a more detailed explanation of the external user schema and internal database schema according to the present invention.

#### User schema

The LDAP V3 attribute schema is extended to include the following information for each attribute. For the keyword MERGETBL, the first item on the keyword list is the relational database (e.g., DB2) table name for the merged table, and the second item on the list is the DB2 column name in the merged table. For the keyword ATTRTBL, the first item on the list is the DB2 table name for the per attribute table, and the second item on the list is the DB2 column name in the attribute table. For the keyword INDEX, the character 'F' in the list represent regular indexes, and the character 'R' in the list represents reverse indexes. Of course, the particular characters used are merely exemplary.

Each attribute can exist either in the merged table, the per attribute table, or both, depending on the characteristics of the given application data. The following is an example of the extended attribute schema

definition for the X.500 surname attribute, which contains the family name of a person:

```
5      (2.5.4.4 NAME 'sn' MERGETBL (merged $ sn)
        ATTRTBL (sn $ sn)
        INDEX (F $ R ))
```

In this example, attribute values "sn" can be found in both the merged table (table name "merged" and column name "sn") and the attribute table (table name "sn" and column name "sn"). Moreover, both regular and reverse indexes are  
10 created for the "sn" column in both tables.

This schema information preferably is included in a distinct configuration file, in this example, a file called slapd.ext.conf. For attributes that do not show up in the extended configuration file, a per attribute table is  
15 created as has been previously described. In a representative embodiment, the merged and per attribute tables are stored in the relational database, which may be DB2. In a DB2 embodiment, if the attribute name is less than 15 characters, the DB2 table name and column name  
20 preferably are the attribute name. If, however, the attribute name is longer than 15 characters, preferably the attribute name is truncated to 15 characters and a two digit counter value is appended at the end to create a table name. If the table name exists, the counter is incremented and an  
25 attempt is then made to create the table. Preferably, the column name is the same as the table name.

## DB2 Schema

The following is the schema for the merged attribute table in the preferred embodiment using DB2 as the relational database backing store.

5 *Merged Attribute Table*

The purpose of the merged attribute table is to store normalized attributes to assist during the search function. As illustrated in **Figure 8**, a column of this table is created when the attribute is a single-value attribute and  
10 the attribute syntax is not binary. If the user does not specify whether the attribute is a single or multi-valued attribute, the merged table creates a new column in the merged table if the MERGETBL keyword appears in the slapd.ext.conf file for this attribute. A reverse column is  
15 created if the INDEX keyword appears in the slapd.ext.conf file for this attribute and the 'R' option is specified. A DB2 index is created for the reversed data column. A DB2 index is created if the INDEX keyword appears in the slapd.ext.conf file.

20 *Attribute table*

The attribute table is created if the attribute does not appear in the slapd.ext.conf file, or if the attribute appears in the slapd.ext.conf file and ATTRTBL keyword is specified. The attribute table is also created when the  
25 attribute value is a truncated value. If the length of the

column is longer than 240 bytes, for example, a truncated column is created for indexing. A reverse column is created if the INDEX keyword appears in the slapd.ext.conf file for this attribute and the 'R' option is specified. A DB2 index  
5 is created for the reversed data column.

A DB2 index also is created if the INDEX keyword appears in the slapd.ext.conf file. In particular, if the max length of the attribute is longer than 240 bytes, for example, the index is created on the truncated column. If  
10 the max length of the attribute is less than 240 bytes, for example, the index is created on the attribute column.

#### LDAP Filter to SQL Translation

One of the main functions of LDAP/DB2 is to translate the LDAP filter an expression into SQLs. A filter  
15 translator (rdbm\_xlfilter.c) is used to generate the equivalent SQL expression corresponding to an LDAP filter that can be used in the WHERE clause of an SQL SELECT statement. The following describes the translation that is performed to generate the SQL expressions. The LDAP filter  
20 translator also generates the list of SQL tables needed for the FROM clause.

#### Base Level Search:

```
25      SELECT entry.EntryData,  
        from ldap_entry as entry  
        where entry.EID in (  
          select distinct ldap_entry.EID  
          from <table list>  
          where (ldap_entry.EID=<root dn id> )  
             <sql where expressions>)
```

One Level Search:

```
SELECT entry.EntryData,  
      from ldap_entry as entry  
5      where distinct ldap_entry.EID  
      from ldap_entry, <table list>  
      ldap_entry as pchild, <list of tables>  
      where ldap_entry.EID=pchild.EID  
      AND pchild.PIED=<root dn id>  
10      <sql where expressions>
```

Subtree Search

```
SELECT entry.EntryData,  
      from ldap_entry as entry  
15      where entry.EID in (  
      select distinct ldap_entry.EID  
      from ldap_entry, ldap_desc, <table list>  
      where  
      (LDAP_ENTRY.EID=ldap_desc.DEID AND  
20      ldap_desc.AEID=<root dn id>)  
      ldap_entry as pchild, <table list>  
      where ldap_entry.EID=ldap_desc.EID  
      AND ldap_desc.AEID=%d <where expressions>).
```

25 In the above representation, <table list> and <where  
expression> are the two null terminated strings returned by  
the SQL generator. The <root dn id> is the unique  
identifier of the root dn. The where clause should only be  
generated if <where expression> is not the empty string and  
30 no errors where detected in the parsing the LDAP filter.

The following is the detailed description of the LDAP  
filter to SQL translation rules. In the translation rules,  
the attr\_tablename is the attribute table for the specified  
attribute and attr\_columnname is the column name containing  
35 the attribute values. The tablename is the name of the  
merged table. The column name is the column name containing

the attribute values. The filter translator invokes an attr\_get\_info function that returns the rdbm\_attr\_info data structure, which contains the fully qualified SQL tables name and column name corresponding to the specified attribute name.

EQUALITY (=) translation rule:

If two tables exists in the rdbm\_attr\_info data structure, the equality type LDAP filter "(attribute=value)" is translated to:

```
10 (SELECT EID FROM attr_tablename WHERE attr_columnname =  
'value') UNION  
(SELECT EID FROM tablename WHERE columnname = 'value').
```

If only one table exists in the rdbm\_attr\_info data structure, the equality type LDAP filter "(attribute=value)" is translated to:

```
15 (SELECT EID FROM tablename WHERE columnname = 'value').
```

GREATER or EQUAL (>=) translation rule:

If two tables exists in the rdbm\_attr\_info data structure ,the greater or equal type LDAP filter "(attribute>=value)" is translated to

```
20 (SELECT EID FROM attr_tablename WHERE attr_columnname >=  
'value') UNION  
(SELECT EID FROM tablename WHERE columnname >= 'value').
```

If only one table exists in the rdbm\_attr\_info data structure, the equality type LDAP filter "(attribute>=value)" is translated to:

```
25 (SELECT EID FROM tablename WHERE columnname >= 'value').
```

LESS or EQUAL (<=) translation rule:

If two tables exists in the rdbm\_attr\_info data structure, the less or equal type LDAP filter "(attribute>=value)" is translated to

5 (SELECT EID FROM: attr\_tablename WHERE attr\_columnname <= 'value') UNION  
(SELECT EID FROM: tablename WHERE columnname <= 'value').

If only one table exists in the rdbm\_attr\_info data structure, the equality type LDAP filter

10 "(attribute<=value)" is translated to:

(SELECT EID FROM tablename WHERE columnname <= 'value').

SUBSTRING (\*) translation rule:

If two tables exists in the rdbm\_attr\_info data structure, the sub-string type LDAP filter

15 "(attribute=value-with-stars)" is translated to

(SELECT EID FROM attr\_tablename WHERE attr\_columnname LIKE 'value with percents') UNION  
(SELECT EID FROM tablename WHERE columnname LIKE 'value with percents').

20 If only one table exists in the rdbm\_attr\_info data structure, the equality type LDAP filter "(attribute=value-with-stars)" is translated to:

(SELECT EID FROM tablename WHERE columnname LIKE 'value with percents').

25 PRESENCE translation rule:

If two tables exists in the rdbm\_attr\_info data structure, the sub-string type LDAP filter "(attribute=\*)" is translated to

```
5 (SELECT EID FROM attr_tablename WHERE attr_columnname IS NOT
  NULL) UNION
  (SELECT EID FROM tablename ).
```

If only one table exists in the rdbm\_attr\_info data structure, the presence type LDAP filter "(attribute=\*)" is translated to:

```
10 (SELECT EID FROM tablename).
```

Approximate Search rule:

If two tables exists in the rdbm\_attr\_info data structure, the sub-string type LDAP filter "(attribute~=value)" is translated to

```
15 (SELECT EID FROM attr_tablename WHERE
  SOUNDEX(attr_columnname)=SOUNDEX('value')) UNION
  (SELECT EID FROM tablename WHERE
  SOUNDEX(columnname)=SOUNDEX('value')).
```

If only one table exists in the rdbm\_attr\_info data structure, the equality type LDAP filter "(attribute~=value)" is translated to:

```
20 (SELECT EID FROM: tablename WHERE
  SOUNDEX(columnname)=SOUNDEX('value')).
```

where SOUNDEX is the soundex library function provided by  
25 DB2.

The above translation rules can be combined into complex LDAP filters using AND, OR, and NOT operators & or | or !. The AND operator & can be used for the Boolean AND of



any number of LDAP filters which can be simple or complex.  
The OR operator | can be used for the Boolean OR of any  
number of LDAP filters which can be simple or complex. The  
NOT operator ! can be used for the Boolean NOT of a single  
5 LDAP filter that may be simple or complex.

Basically, LDAP logic operator & is translated into SQL  
INTERSECT to intersect results from multiple SELECT  
statements. LDAP operator | is translated into SQL UNION to  
union results from multiple SELECT statements. The LDAP NOT  
10 operator ! is translated into SQL NOT IN to exclude results  
from a select statement. This process is described in more  
detail in copending application Serial No. 09/160,022,  
assigned to the assignee of this application, the disclosure  
of which is incorporated herein by reference.

15 A few examples (using the Nested Select function) are  
given below to show the equivalent SQL expressions that are  
generated for some typical LDAP filters. Because SQL  
parameter markers are used in the query, the question marks  
(?) in the query represent the attribute values in the LDAP  
20 filter.

Filter String:

-----  
(sn=SMITH)

Scenario 1: if "sn" is a single value attribute:

25 slapd config file:  
slapd.at.conf:  
(2.5.4.4 NAME 'sn' SUP name SINGLE-VALUE)  
slapd.ext.conf:

(2.5.4.4 NAME 'sn' MERGETBL (merge \$ sn)  
 SQL Statement: SELECT entry.EntryData, CREATOR, MODIFIER,  
 CREATE\_TIMESTAMP, MODIFY\_TIMESTAMP, entry.EntryBlob,  
 entry.Entrysize FROM LDAP\_ENTRY as entry WHERE entry.EID in  
 5 ( SELECT distinct SSHI.LDAP\_ENTRY.EID FROM  
 LDAP\_ENTRY,ldap\_desc WHERE (LDAP\_ENTRY.EID=ldap\_desc.DEID  
 AND ldap\_desc.AEID=?) AND LDAP\_ENTRY.EID IN (SELECT EID FROM  
 merge WHERE SN = ?)).

10 Scenario 2: if "sn" is a multiple value attribute and two  
 tables are specified in the configuration file.

slapd config file:

slapd.ext.conf:

(2.5.4.4 NAME 'sn' MERGETBL (merge \$ sn)  
 15 ATTRTBL (sn \$ sn))  
 SQL Statement: SELECT entry.EntryData, CREATOR, MODIFIER,  
 CREATE\_TIMESTAMP, MODIFY\_TIMESTAMP, entry.EntryBlob,  
 entry.Entrysize FROM LDAP\_ENTRY as entry WHERE entry.EID in  
 ( SELECT distinct SSHI.LDAP\_ENTRY.EID FROM  
 20 LDAP\_ENTRY,ldap\_desc WHERE (LDAP\_ENTRY.EID=ldap\_desc.DEID  
 AND ldap\_desc.AEID=?) AND LDAP\_ENTRY.EID IN ((SELECT EID  
 FROM merge WHERE SN = ?) UNION  
 (SELECT EID FROM SN where SN= ?))

#### Prototype and Performance Results

25 A prototype has been developed to measure the  
 performance improvements of the inventive database schema.  
 The prototype simulates how the tables are populated in  
 ldap\_add with per attribute tables and the merged table.

For Scenario 1 above, when all indexes include 1036  
 30 entries and 10 attributes, the merged table includes 715  
 secs and the attribute table includes 2221 secs. For  
 Scenario 2, in which no indexes except EID on ldap\_entry are  
 included and 1036 entries and 10 attributes are included,  
 the merged table includes 587 secs and the attribute table  
 35 includes 2081 secs.

Experiments were performed with 9000 entries. Based on these experiments, it was found that the time it takes to populate the merged table was about one third of the per attribute table.

5 As noted above, the invention may be implemented in any hierarchical directory service in which a relational database management system (RDBMS) is used to provide a backing store function. Thus, for example, the principles of the invention may be carried out in an X.500 directory  
10 service or hereinafter-developed LDAP implementations. The SQL query generated according to the present invention is used to access the relational database, and results are then returned in response to this query. The invention may also be implemented within a relational database management  
15 system being used as an add-on to a directory service. One of ordinary skill will appreciate that the invention can be applied to any relational database management system (RDBMS) and not simply DB/2, the implementation described above. Thus, for example, the relational database may be Oracle,  
20 Sybase or any other third party supplied backing store. In addition, the EID sets approach can also be applied to b-tree based LDAP server implementation.

Moreover, although the preferred embodiment has been described in the context of generating a Structured Query  
25 Language (SQL) query, the inventive technique should be

broadly construed to extend to any relational database query language.

One of the preferred embodiments of the routines of this invention is as a set of instructions (computer program  
5 code) in a code module resident in or downloadable to the random access memory of a computer.

Having thus described our invention, what we claim as new and desire to secure by Letters Patent is set forth in the following claims.

006372.00197:0416317.01